A Neuro-Symbolic Approach to Runtime Optimization in Resource Constrained Heterogeneous Systems

Chitra Subramanian[1], **Sarath Swaminathan**[1], **Miao Liu**[1], **Mauricio Longinos**[2], **Aporva Amarnath**[1], **Karthik Swaminathan**[1], **Martin Cochet**[1], **Kevin Fernández**[2] and **Pradip Bose**[1]
[1]IBM Research
[2]IBM CIO Office
Correspondence: cksubram@us.ibm.com

Abstract

Our approach to runtime optimization of heterogeneous systems using Reinforcement Learning (RL) and neuro-symbolic Logical Neural Networks (LNN) is described. We provide details of a method for optimizing the runtime allocation of power and processor resources in Heterogeneous System-on-Chip (HSoC) applications to maximize the performance/watt. We demonstrate promising results in creating human interpretable and interactable AI policy, learning with less data by incorporating domain knowledge, and use of logical reasoning to extend the optimization window beyond the training set. HSoC register-transfer level (RTL) simulation results are used to show the comparative advantage of the neuro-symbolic policy implementation versus state-of-the-art hardware power management.

1 Introduction

Runtime optimization in resource constrained heterogeneous systems is an important research area across multiple domains. Applications include real-time scheduling with system constraints in High Performance Computing (HPC), Cloud and Data centers, Heterogeneous System-on-Chip (HSoC); routing in telecom networks, autonomous vehicles; process control of industrial Digital Twins; and more. For example, the main challenge for enabling parallel programming models in next-generation heterogeneous multi-core SoCs is in jointly optimizing both performance and power efficiency. Standard heuristic rule-based scheduling and power management techniques are inefficient in these applications due to the wide range of workflow requirements to support; variable configurations, power-frequency characteristics; and stochasticity in task arrival times, processing times, network congestion and deadlines. Recent work using AI methods of Reinforcement Learning (RL) and Deep Neural Network (DNN) based policy have shown promise in improving operating efficiency [Mandal et al., 2019; Zuckerman et al., 2021]. However, these approaches share the common drawbacks of deep learning methods, such as, requiring very large number of training examples; rules learned cannot be extended to cases much beyond training examples; large model sizes resulting in response times that are too slow for real-time control; and a lack of human interpretability in the decisions.

1.1 Neuro-Symbolic Approach

To address the challenges described above and meet the disparate runtime optimization requirements of heterogeneous systems, RL frameworks that utilize distributed intelligence with the ability to learn hybrid neuro-symbolic policies, need to be explored. The key to architecting this RL system lies in the neuro-symbolic policy learning framework used within each component (agent) of the distributed intelligence. The recently proposed neuro-symbolic Logical-Neural-Networks (LNNs) [Riegel et al., 2020; Chaudhury et al., 2021] provide a constrained optimization framework, where the neurons behave as logical operatives without enforcing a specific structure for the learning function. Choosing LNNs as the differentiable function approximators for rule induction allows the learned policy to be interpretable and to easily incorporate expert domain knowledge into the rule set. The ability of LNNs to generate compact rules-based light weight policies, unlike previous DNN based RL methods, enables fast response times. In addition, LNNs allow a distributed intelligent multi-agent system design by enabling logical inference and predictive action based on partial observability. In safety critical applications, RL policy based on LNNs can incorporate valuable features such as: guard rail safety constraints formulated as symbolic representations; robust efficient model training and optimization for uncertain environments and rare events; and interpretable AI model decisions permitting system control with human-in-the-loop.

1.2 Heterogeneous System-on-Chip Application

We use the example of a Heterogeneous SoC (HSoC) application to illustrate a scalable method of performance and power efficient runtime optimization using multi-agent RL framework with LNN policy advisor. Figure 1 shows a heterogeneous SoC with a diversity of processor tiles and a potential workflow graph that needs to be executed upon it. Processing element (PE) tiles with similar functionality are depicted in the same colors. The optimization challenge illustrated in this example is to maximize the throughput of the workflow graphs for random arrivals and deadlines, given a constrained system power envelope. This requires optimizing

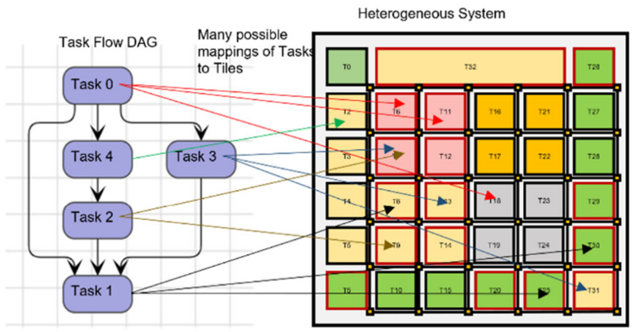


Fig 1: Illustrative example of scheduling workloads to HSoC with constraints.

the task scheduling to the appropriate processor tiles as well as optimizing the power sharing between the tasks.
Therefore, the objective of this HSoC optimization problem is to determine the allocation of power and processor resources at runtime to maximize the system performance/watt. Scheduling tasks to processing element (PE) tiles of a HSoC requires response times of a few milliseconds, with a few hundred micro-second power management (PMT) response time for allocating optimal power to the PE tiles. We propose that a multi-agent neuro-symbolic reinforcement learning system that learns compact logical rules would be necessary for an efficient control system that meets these tight timing requirements. As the first step in developing such a system, the goal of our present work is to demonstrate LNN rule optimized power sharing between PE tiles to maximize job throughput provided the PE task assignments at runtime from an external scheduler. In section 2 we discuss the problem setup including the formulation, system variables, and RL training implementation for learning optimal action. In section 3 we provide results including details of LNN rule learning using RL, the resulting interpretable temporal neuro-symbolic rules and simulated runtime inference including for a state-of-the-art HSoC Integrated Circuit (IC) chip RTL.

2 Optimization Framework

2.1 Problem Formulation and System Variables

The HSoC is organized as a grid of PE tiles as in Figure 1, and the goal is to maximize job throughput for a fixed maximum allowed HSoC power envelope, given the random arrival of jobs at runtime. Each job is composed of several tasks that are connected by a directed acyclic workflow graph (DAG) as shown in the example of Figure 1. At runtime, an external scheduler assigns the incoming queued up tasks to the fastest available PE tiles. Tasks which execute in parallel on the HSoC, such as Task4 and Task3 in the example DAG, need to share the total available power to stay within the maximum power constraint. The task completion time on any PE tile has a non-linear inverse relationship to the power provided, determined by its frequency vs power characteristics. To maximize job throughput, the power sharing between PEs needs to be optimized so that tasks are efficiently processed on their assigned PEs to minimize the job execution time, i.e., the DAG makespan. The PE power sharing allocation as a

Variability/Stochasticity	Property
Design time	Configuration of cores Workflow DAGs
Random run-time	DAG arrival times DAG Completion deadlines
Complex dependencies run-time	Processor assignment Power assignment Congestion Temperature Task completion time
Complex dependencies lifetime	Wear out

Table 1: Sources of variation and their dependencies in Heterogeneous System-on-Chip operation.

percent of maximum allowed HSoC power is referred to as 'Tokens'.

There are several variables in this system with design time, runtime, and lifetime variability. Table1 shows these different sources of system variation, categorized by their timeframes and stochastic behaviors. Given the many degrees of uncertainty, the training set required to capture all possible system states for any purely neural (DNN) model is, therefore, explosively large. Neuro-symbolic LNN model's ability to learn while incorporating domain expert rules and the ability to use logical reasoning during inference time are vital to reducing the training set and model size to make the optimization solution more tractable.

Our goal was to demonstrate the LNN capability to learn temporal rules that are human interpretable and to obtain an RL policy that is optimized for PE power sharing at runtime. We therefore included the following critical factors: random runtime variations in DAG arrival times and completion deadlines, variable PE assignments to tasks based on a scheduler, and model-based variation of task completion time vs power assigned for the PEs. In this first RL LNN study we considered the design time variables to be a finite set to work with and did not include any lifetime wear out.

2.2 RL Implementation

The LNN rule learning was done using an RL Gym Environment. Figure 2 shows the setup for RL LNN rule learning, where the PE Tile agent learns optimal power sharing rules based on variable DAGs and PE task assignment schedules from an external scheduler. Each PE Tile has an independent

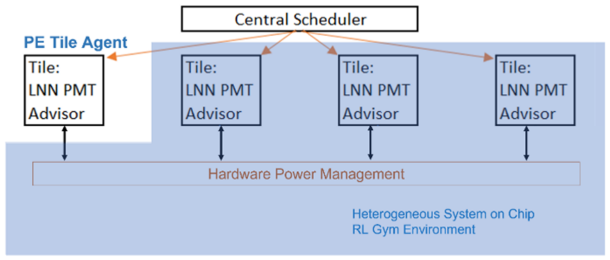


Figure 2: Setup for RL LNN rule learning to learn optimal PE Tile power sharing rules.

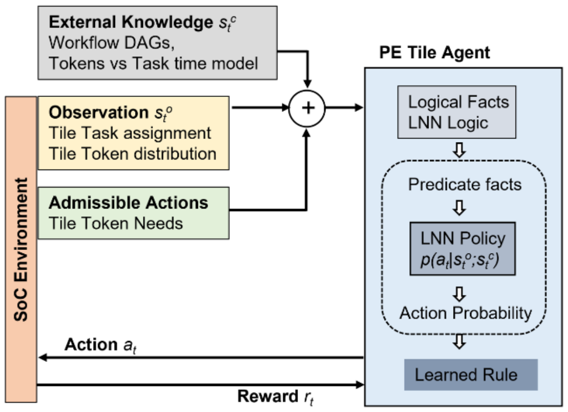


Figure 3: RL-based LNN rule learning system for HSoC power token sharing optimization.

LNN power management (PMT) advisor that learns by performing the action of requesting a discrete percentage of the maximum HSoC power (Tokens) at every time step. Since the PE Tile agents are independent and don't interact with each other, the learning was done for one PE Tile agent at a time, considering all the other PE Tile agents to be part of the HSoC environment for that purpose. In this setup, all the PE Tile agents ultimately learn fully compatible LNN PMT policies. The system relied upon an existing underlying 'Hardware Power Management' unit designed to actuate low-latency PE power sharing based on state-of-the-art distributed power management techniques [Shah et al., 2021]. However, this hardware PMT unit was abstracted with ideal characteristics for purposes of this LNN temporal rule learning setup.

Figure 3 shows the implementation of the LNN rule learning system using the RL Gym environment. As described in Figure 2, the SoC Environment (SoC Env) is designed to mimic the behavior of a simplified heterogeneous System-on-Chip where one PE Tile is managed by the agent and all other tiles by the environment. Each training episode consists of completing a single job or workflow DAG. The SoC Env keeps track of the task assignment, progress of tasks, and power token distribution between tiles during the lifetime of a job. At each time step the PE Tile agent takes the action of allocating a discrete percentage of maximum allowed power for itself. Reinforce algorithm and dense reward system with the reward function based on completion time of all the tasks were used for training. These aspects follow any standard RL policy training setup. The changes made for LNN-based RL training are: (i) use of logical predicates and observations; (ii) incorporating domain expert knowledge; (iii) control of admissible actions to impose guard rails. Examples of logical predicates include ones that define whether a task is assigned to the agent's PE Tile ('TaskAssigned'), whether any parallel tasks exist at that time step ('SiblingTasks'), and whether parent tasks have completed ('ParentTaskCompleted'). Expert knowledge about the features of the task within the DAG, such as its contribution to the makespan critical path and presence or absence of other concurrent tasks, are provided during training as an additional real valued predicate ('Slack'). Guard rail rules take care of edge cases such as forbidding the agent from taking the action of allocating 0% power to itself while processing a task or taking all 100% power while a sibling task is active. These 3 aspects allow greater human interaction and control to reduce training time of LNN based policy and produce fully interpretable rules.

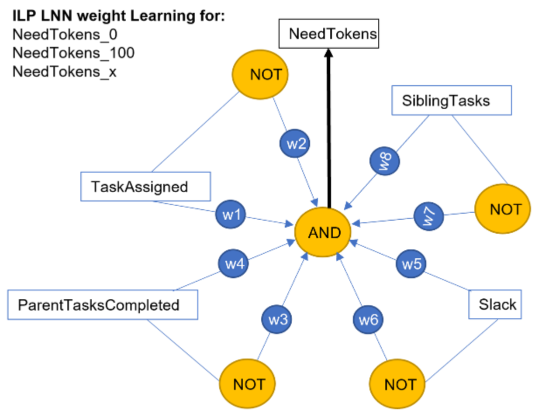


Figure 4: LNN rule learning template using weighted input predicates, Łukasiewicz logical conjunction and negation.

3 Results and Discussion

3.1 Learning LNN Model-Based Policy

The goal of the learned LNN policy was to determine the percent share of the max power that the PE Tile agent should allocate for itself to minimize the DAG makespan. The continuum of all possible values of power between 0% and 100%

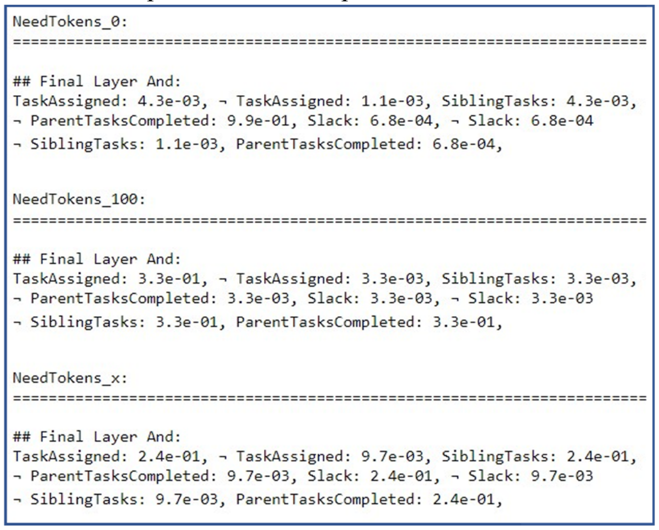

```
NeedTokens_0:
======================================================================

## Final Layer And:
TaskAssigned: 4.3e-03, ¬ TaskAssigned: 1.1e-03, SiblingTasks: 4.3e-03,
¬ ParentTasksCompleted: 9.9e-01, Slack: 6.8e-04, ¬ Slack: 6.8e-04
¬ SiblingTasks: 1.1e-03, ParentTasksCompleted: 6.8e-04,


NeedTokens_100:
======================================================================

## Final Layer And:
TaskAssigned: 3.3e-01, ¬ TaskAssigned: 3.3e-03, SiblingTasks: 3.3e-03,
¬ ParentTasksCompleted: 3.3e-03, Slack: 3.3e-03, ¬ Slack: 3.3e-03
¬ SiblingTasks: 3.3e-01, ParentTasksCompleted: 3.3e-01,


NeedTokens_x:
======================================================================

## Final Layer And:
TaskAssigned: 2.4e-01, ¬ TaskAssigned: 9.7e-03, SiblingTasks: 2.4e-01,
¬ ParentTasksCompleted: 9.7e-03, Slack: 2.4e-01, ¬ Slack: 9.7e-03
¬ SiblingTasks: 9.7e-03, ParentTasksCompleted: 2.4e-01,
```

Figure 5: Learned weights for power token sharing rules corresponding to each of its input predicates.

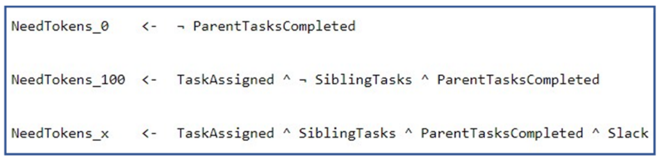

```
NeedTokens_0    <-   ¬ ParentTasksCompleted


NeedTokens_100  <-   TaskAssigned ^ ¬ SiblingTasks ^ ParentTasksCompleted


NeedTokens_x    <-   TaskAssigned ^ SiblingTasks ^ ParentTasksCompleted ^ Slack
```

Figure 6: Code output of RL learning showing interpretable rules for power token sharing needs of a PE Tile agent.

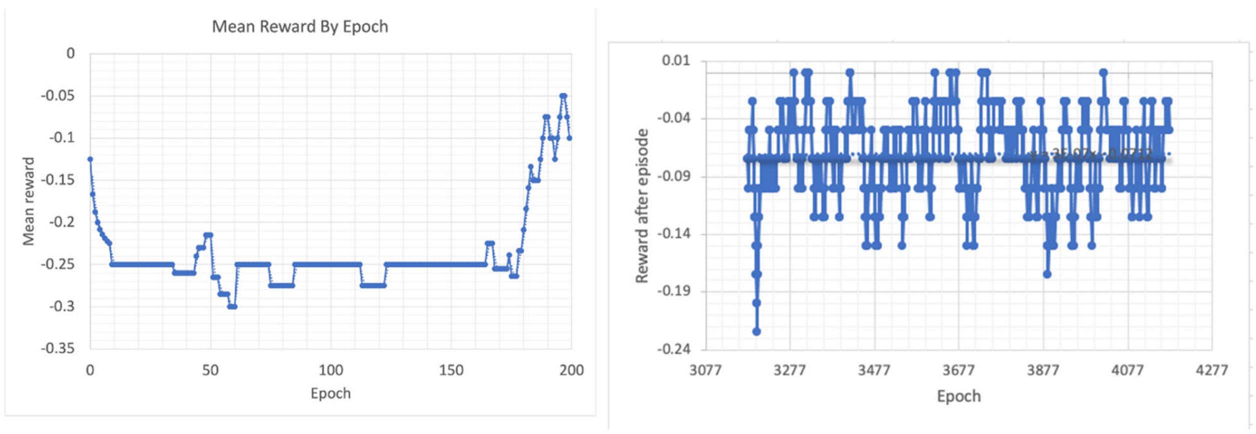


Figure 7: Plots of the rewards during LNN RL training as a function of training epochs. The rewards attain steady state values within 200 epochs (left) that match the values after several thousand epochs (right), indicating training completion in a very short time.

was discretized into 10 equal intervals and the policy would select one of them to take as the action at each time step. Distinct LNN rules were learned simultaneously during training, i.e, NeedTokens_0 for 0% of power to the PE Tile, NeedTokens_100 for 100% of power share, and NeedTokens_x for x% of power share where $0 < x < 100$. A Łukasiewicz logical conjunction with weighted input predicates and negated predicates was defined as the operator for LNN rule learning as shown in Figure 4. Inductive Logic Programming (ILP) along with double-description method of LNN rule learning [Sen et al., 2022] was used to learn interpretable lifted First Order Logic LNN rules. The weights learned are shown in Figure 5. A weight threshold of 0.1 was used to filter out the irrelevant predicates for each rule statement, resulting in the final rules as shown in Figure 6.

The learned rules are straight forward to interpret. The PE Tile agent allocates 0% power to the PE whenever its 'ParentTasksCompleted' predicate is NOT observed to be logically True. This can occur either because the PE is waiting for the parent tasks of its assigned task to complete, or it has not been assigned a task and is idle. Under both conditions the PE Tile is in standby mode and needs to be allocated 0% power, hence the learned rule for NeedTokens_0. On the other hand, when a task is assigned to a PE Tile and its parent tasks have completed, it needs to have power allocated for task execution. If there are no sibling tasks to run in parallel, all available HSoC power needs to be allocated to the PE for minimizing the DAG makespan, hence the NeedToken_100 rule. However, when sibling tasks exist, the NeedTokens_x rule is activated, and the 'Slack' predicate determines the optimal share of power to the PE Tile. As previously stated, the 'Slack' predicate takes on values based on pre-processed expert information for the assigned task's features in the context of its DAG. These 'Slack' predicate values range between 0 and 1 and determine the probabilistic activation for different x% power selections. Therefore, the 'Slack' predicate can be viewed as a family of activation curves for different x% power allocations that the RL system can learn to choose from under different system conditions.

We have thus successfully demonstrated the ability to learn human interpretable LNN RL policy, incorporating domain expert knowledge, and guard rail control of admissible actions to minimize training time. Figure 7 shows the RL rewards tracking versus training epoch. This shows steady state rewards at only 200 epochs indicating quick LNN rule learning and the extendibility to unseen examples will be illustrated by the inference results next.

3.2 Simulated Runtime Inference Results

Table 2 contains the runtime job information provided by the scheduler to the LNN power management agent within each PE Tile. The scheduling information for a single 6-task-DAG

Task type	id	type	current service time	dag type	dag id	task tid	task priority	dag dtime	task sub deadline	task parent ids	task arrival time	current job start time	current job end time
fft_256	10	fft_accel	10	5	517	0	1	5370	3220		0	0	10
fft_64	8	gpu	80	5	517	4	1	5370	160	0	10	10	90
cnn_14	10	fft_accel	180	5	517	3	1	5370	1610	0	10	10	190
decoder	8	gpu	20	5	517	2	1	5370	1440	4	90	90	110
cnn_32	8	gpu	50	5	517	1	1	5370	590	0 2 3	190	190	240
cnn_14	10	fft_accel	10	5	517	5	1	5370	1610	1	240	240	250

Table 2: External scheduler input provided to the LNN power management advisor in every PE Tile.

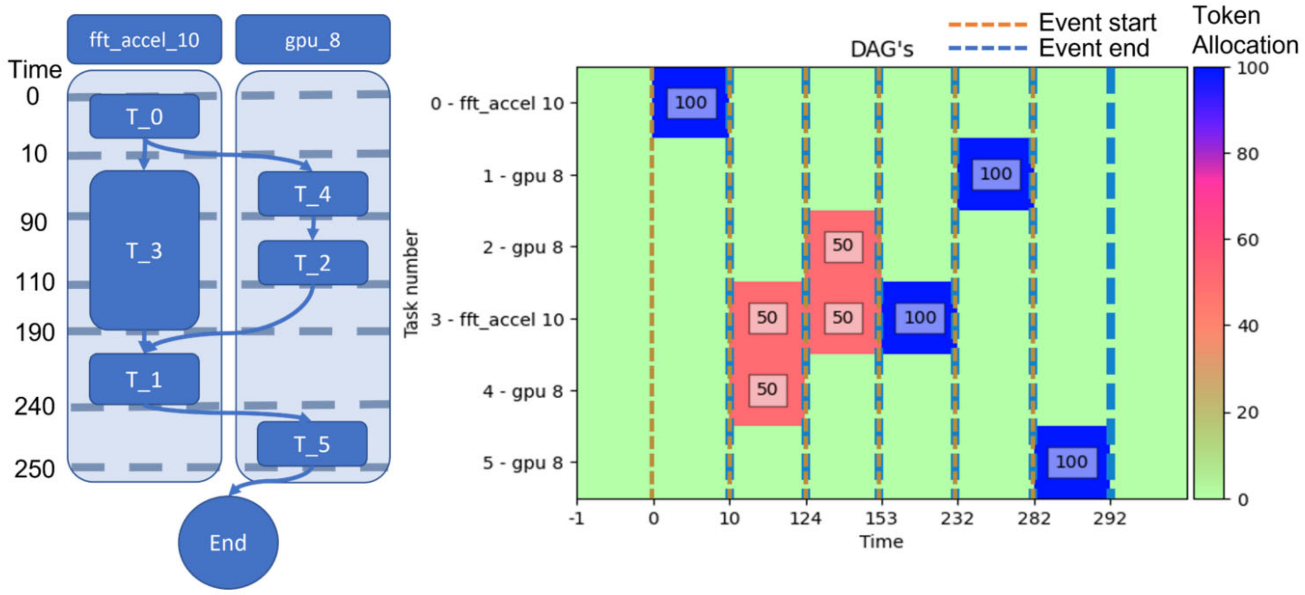


Figure 8: Example 6-task-DAG (left) shows event times when the power sharing decisions need to be made if every task was assigned full 100 power tokens. The Gantt chart (right) shows the simulated current default Hardware Power Management sharing power tokens equally between parallel tasks and the resulting slowdown in task completion times.

example is shown in the table. The 'task tid' column shows the task number within the DAG and 'task type' column provides the task description. The 'type' and 'id' columns show the assignment PE type and the PE Tile identification. The 'current service time' column provides the execution time of the task assigned if the PE Tile is allocated maximum power. The 'dag type' column has the DAG identification number from the finite set of predetermined DAGs and the 'dag id' is the job number. The 'dag dtime' is the DAG completion deadline requirement and the 'task sub deadline' is the resulting pre-calculated task deadline requirement that is used to calculate the 'Slack' predicate values. A single 'Slack' predicate based on a criterion that maintains the pre-calculated ideal ratio of 'task sub deadline' between parallel sibling tasks was used for this illustration. The power sharing ratio selected by the LNN policy therefore tends to provide more power to speed up the sibling task assigned to a slower PE while providing less power to the sibling that is assigned to a fast PE in order to maintain the ideal 'task sub deadline' ratio and minimize DAG makespan. This is the domain knowledge input indicated by the 'Slack' predicate. The interpretation of 'task parent ids', 'task arrival time', 'current job start time' and 'current job end time' are straightforward. The 'task priority' information is not currently being used but is available for future use.

At any given time step the input table only contains the information for rows that were recently executed and the task to PE assignments for a limited look ahead time step and not for the entire DAG to reduce data storage and communication. The PE Tile LNN PMT advisor makes runtime power allocation decisions at event time points at the start and completion of any task in the system. For example, in the 6-task-DAG of Figure 8 (left) with fft_accel_10 and gpu_8 as PE assignments, the PMT decision event time points, assuming all tasks are assigned full 100 tokens, shows a DAG completion time of 250 cycles. The current Hardware Power Management unit shares the maximum allowed HSoC power equally among all the active PE Tiles without considering the impact

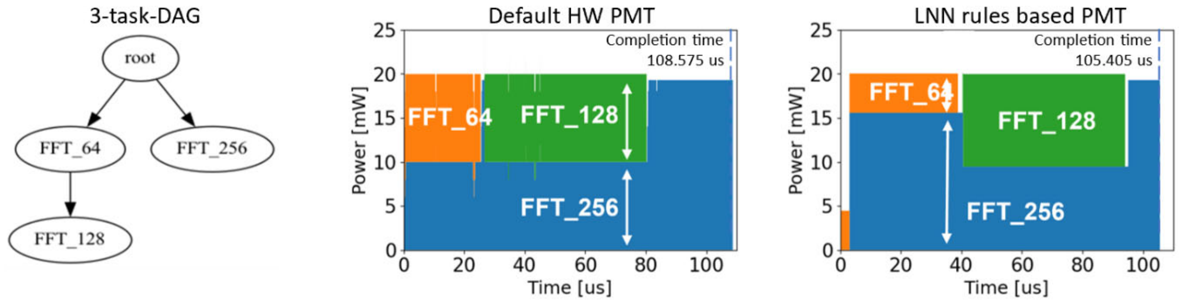
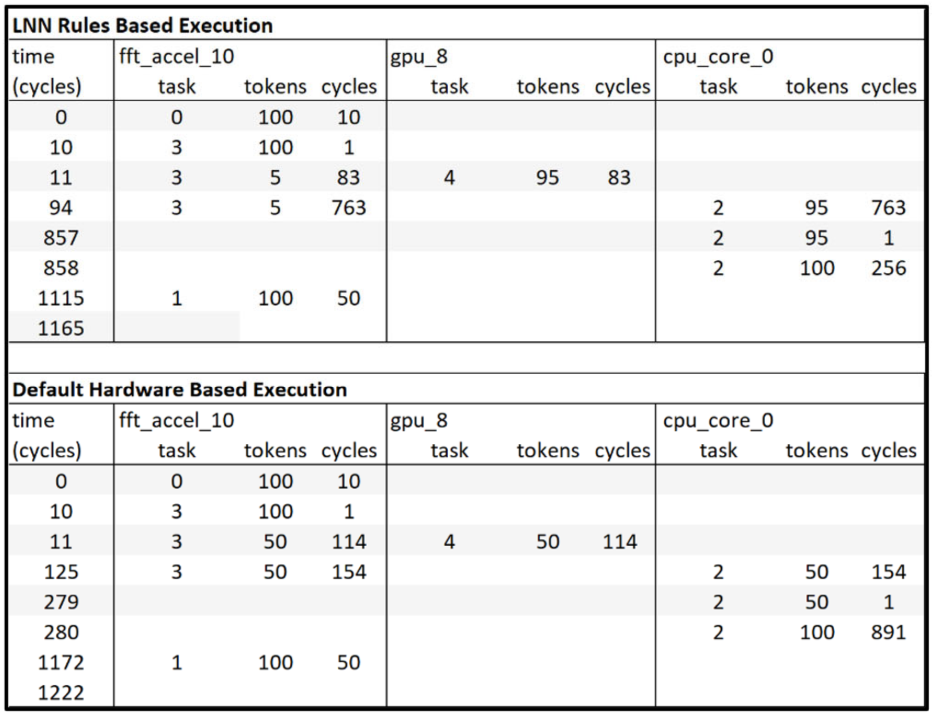


Figure 9: Hardware RTL simulations of a 3-task-DAG (left) and its power sharing implementation vs time for default hardware (middle) and LNN rule based (right).



LNN Rules Based Execution

time (cycles)	fft_accel_10 task	fft_accel_10 tokens	fft_accel_10 cycles	gpu_8 task	gpu_8 tokens	gpu_8 cycles	cpu_core_0 task	cpu_core_0 tokens	cpu_core_0 cycles
0	0	100	10						
10	3	100	1						
11	3	5	83	4	95	83			
94	3	5	763				2	95	763
857							2	95	1
858							2	100	256
1115	1	100	50						
1165									

Default Hardware Based Execution

time (cycles)	fft_accel_10 task	fft_accel_10 tokens	fft_accel_10 cycles	gpu_8 task	gpu_8 tokens	gpu_8 cycles	cpu_core_0 task	cpu_core_0 tokens	cpu_core_0 cycles
0	0	100	10						
10	3	100	1						
11	3	50	114	4	50	114			
125	3	50	154				2	50	154
279							2	50	1
280							2	100	891
1172	1	100	50						
1222									

Table 3: Simulated progression of 5-task-DAG (shown in Fig.1) using LNN power sharing vs default hardware.

on task completion times and DAG makespan as shown in Figure 8 (right). Since the reduced power allocation per task results in slower completion times on tasks T_2, T_3, and T_4, the overall DAG with power sharing takes an additional 42 cycles to complete at 292 cycles.

The simulated execution and power sharing selection for learned LNN rules based PMT versus the current default hardware PMT of the 5-task-DAG of Figure 1, is shown in Table 3. In this case, based on the scheduler assignments, Task 0 executes on the fft_accel_10 PE; followed by Tasks 3 and 4 executing in parallel on the fft_accel_10 and gpu_8 PEs respectively; followed by Tasks 3 and 2 executing in parallel on the fft_accel_10 and cpu_core_0 PEs respectively; and finally Task 1 executing on fft_accel_10 PE to complete the DAG. The optimized power sharing decisions using the LNN PMT case results in a better DAG makespan of 1165 cycles relative to current hardware PMT which required 1222 cycles to complete the DAG. Eliminating minor single cycle PMT delays seen during execution at event time points (such as cycles 10, 279 and 857) would produce a more accurate comparison. However, the magnitude of improvement in DAG makespan is dependent on the nature of the DAG, the component task execution time characteristics, and the assigned PEs. Therefore, the results in Table 3 should only be considered qualitatively as demonstrated proof of concept for advantage of LNN PMT over current state-of-the-art hardware PMT technique.

Additionally, we verified LNN PMT advantage in DAG makespan over the current hardware PMT using RTL level simulations of our recent HSoC IC chip for a 3-task-DAG as shown in Figure 9. Setting the maximum allowed IC power consumption to 20mW, the default hardware PMT allocated the same 10mW of power at the start to the parallel tasks assigned to FFT_64 and FFT_256 PEs. In contrast, the LNN rules based PMT allocated 15mW to the task executing on FFT_256 PE and the remaining 5mW to the task executing on FFT_64 PE. The LNN rules based PMT resulted in earlier DAG completion time of 105.405usec whereas the default hardware PMT required 108.575usec. As before, these results represent proof of concept validation only and exact magnitude of LNN improvements depend on the DAG and PE characteristics. Also, the minor delays seen at event edges in the LNN PMT simulation can be eliminated through better design optimization.

4 Summary

We have demonstrated promising results in the application of neuro-symbolic techniques to power management of heterogeneous systems as shown in this System-on-Chip use case. We have successfully shown for the first time: (i) Human-interpretable temporal rule learning for real-time optimization using neuro-symbolic LNN; (ii) Incorporation of domain knowledge and guard rail constraints to minimize training; (iii) Learned logical rules that allow operation extension well beyond the training set. While we have only been working with toy example DAGs so far to illustrate these advantages of neuro-symbolic methods, we are currently working on real application workload demonstrations. Our plans include applying probabilistic predictive action and joint scheduling and power optimization using neuro-symbolic methods next.

Acknowledgements

This research was developed with funding from the Defense Advanced Research Projects Agency (DARPA). The views, opinions and/or other findings expressed are those of the authors and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government. Distribution Statement A: Approved for Public Release, Distribution Unlimited.

References

[Chaudhury et al., 2021] S. Chaudhury, P. Sen, M. Ono, D. Kimura, M. Tatsubori, A. Munawar, Neuro-Symbolic Approaches for Text-Based Policy Learning, Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing.

[Mandal et al., 2019] S. K. Mandal, G. Bhat, C. A. Patil, J. R. Doppa, P. P. Pande and U. Y. Ogras, Dynamic Resource Management of Heterogeneous Mobile Platforms via Imitation Learning, IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 27, no. 12, pp. 2842-2854, Dec. 2019.

[Riegel et al., 2020] R. Riegel, A. Gray, F. Luus, N. Khan, N. Makondo, I. Y. Akhalwaya, H. Qian, R. Fagin, F. Barahona, U. Sharma, S. Ikbal, H. Karanam, S. Neelam, A. Likhyani, S. Srivastava, Logical Neural Networks, arXiv:2006.13155v1 (2020).

[Sen et al., 2022] P. Sen, Breno W. S. R. de Carvalho, R. Riegel, A. Gray, Neuro-Symbolic Inductive Logic Programming with Logical Neural Networks, AAAI, 2022.

[Shah et al., 2021] P. Shah, R. G. Shenoy, V. Srinivasan, P. Bose, A. Buyuktosunoglu, TokenSmart: Distributed, Scalable Power Management in the Many-Core Era, IEEE Computer Architecture Letters, Vol. 20, No. 1, Jan 2021.

[Zuckerman et al., 2021] J. Zuckerman, D. Giri, J. Kwon, P. Mantovani, L. P. Carloni, Cohmeleon: Learning-Based Orchestration of Accelerator Coherence in Heterogeneous SoCs, 54th IEEE/ACM International Symposium on Microarchitecture (MICRO 2021).